OpenMP supported features in the UpScale SDK

v1.0, 31st January 2017

ROYUELA, Sara
QUIÑONES, Eduardo
PINHO, Luis Miguel

Table of contents

Introduction

This document presents the OpenMP constructs which can be used when developing applications with the UpScale SDK for the Kalray MPPA processor. Note that individual components of the SDK may support more features when using independently – these are not listed here and should be checked in the components' own documentation.

The model considered in the UpScale implementation for the Kalray MPPA is an accelerator model, where applications start executing in the IO cores of the processor, and offload parallel computation to the computing clusters. Therefore, the support is different in both IO and computing clusters.

For more details on the semantics and usage of the directives and clauses listed in this document, consult the OpenMP specifications.

Support on the IO side

Device constructs.

Device constructs form the accelerator model of OpenMP.

Declare target directive. This directive specifies that variables and functions are mapped to a device.

The syntax is as follows:

```
#pragma omp declare target new-line
declaration-definition-seq
#pragma omp end declare target new-line
```

Target construct. This construct maps variables to a device data environment and executes the construct on that device.

The syntax is as follows:

```
#pragma omp target [clause[ [,] clause] ... ] new-line
structured-block
```

where *clause* is one of the following:

```
device(integer-expression)
map([[map-type-modifier[,]] map-type: ] list)
priority(pryiority-value)
```

Support on the Cluster side

Parallel construct

Parallel construct. This construct starts parallel execution.

The syntax is as follows:

```
#pragma omp parallel [clause[ [,] clause] ... ] new-line
structured-block
```

where *clause* is one of the following:

```
if(scalar-expression)
num_threads(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
reduction(reduction-identifier : list)
```

Worksharing Constructs

A worksharing construct distributes the execution of the associated region among the members of the team that encounters it. The supported worksharings are:

Loop construct. The loop construct specifies that the iterations of one or more associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.

The syntax is as follows:

```
#pragma omp for [clause[ [,] clause] ... ] new-line
for-loops
```

where *clause* is one of the following:

```
private(list)
firstprivate(list)
lastprivate(list)
reduction(reduction-identifier : list)
schedule([modifier [, modifier]:]kind[, chunk_size])
nowait
```

Single construct. The single construct specifies that the associated structured block is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task.

The syntax is as follows:

`#pragma omp single` *[clause[[,] clause] ...] new-line*
structured-block

where *clause* is one of the following:

`private`(list)
`firstprivate`(list)
`copyprivate`(list)
`nowait`

Tasking constructs

Tasking constructs, used combined with master and synchronization constructs, form the tasking model of OpenMP.

Task construct. The task construct defines an explicit task.

The syntax is as follows:

#pragma omp task *[clause[[,] clause] ...] new-line*
structured-block

where *clause* is one of the following:

`if`*(scalar-expression)*
`final`*(scalar-expression)*
`untied`
`default`(`private` | `firstprivate` | `shared` | `none`)
`mergeable`
`private`*(list)*
`firstprivate`*(list)*
`shared`*(list)*
`depend`*(dependence-type : list)*

Master and Synchronization Constructs

Master construct. The master construct specifies a structured block that is executed by the master thread of the team.

The syntax is as follows:

`#pragma omp master` *new-line*
structured-block

Critical construct. The critical construct restricts execution of the associated structured block to a single thread at a time.

The syntax is as follows:

`#pragma omp critical` *[(name)] new-line*
structured-block

Barrier construct. The barrier construct specifies an explicit barrier at the point at which the construct appears.

The syntax is as follows:

```
#pragma omp barrier new-line
```

Taskwait construct.

The taskwait construct specifies a wait on the completion of child tasks of the current task.

The syntax is as follows:

```
#pragma omp taskwait new-line
```

Atomic construct. The atomic construct ensures that a specific storage location is accessed atomically, rather than exposing it to the possibility of multiple, simultaneous reading and writing threads that may result in indeterminate values.

The syntax is as follows:

```
#pragma omp atomic [ read | write | update | capture ] new-line
expression-stmt
```